

Packet-level Simulations of the Flash Worm and the Compact Flash Worm *

Mohamed Abdelhafez
George F. Riley

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
{mofta7,riley}@ece.gatech.edu

November 29, 2011

Abstract

The speed of the spread of Internet worms increased dramatically in recent years. Moreover it is certain that it will continue to increase in the near future with the increase in available bandwidth and network resources. The fastest known worm (only in literature) is the Flash worm, which can infect over a million hosts in less than one second. This paper presents a variant of the Flash worm that we term Compact Flash or CFlash that is capable of spreading even faster than its predecessor. We perform a comparative study between the Flash worm and the CFlash worm using a full packet-level simulator, and the results show the increase in propagation rate of the new worm given the same set of parameters.

1 Introduction

Internet worms have demonstrated that they are serious security risks in recent years; the Code Red worm attack in 2001 infected 360,000 hosts in 14 hours [7]. The direct costs of recovering from this epidemic (including subsequent strains of Code Red) have been estimated to be in excess of \$2.6 billion [8]. The Slammer worm outbreak in January 2002 infected 90% of its vulnerable hosts (75,000) in less than 10 minutes [6], the estimated loss is about \$1 billion [17]. In August 2003 the blaster worm was estimated to have infected more than 500,000 systems worldwide and the cost to North American companies was \$1.3 billion [17]. However, a more recent report showed that the number of infections was between 8 million and 16 million systems [12], marking the blaster worm as the most widely spread worm to date. The Witty worm in 2004 had a malicious payload that targets firewalls; not only did it spread to additional hosts, but it also formatted a portion of the hard drive of the infected host [13].

This increased security risks have caused several researchers to study the behavior of the worms and their spread patterns. Several proposed models have been developed to represent the worm spread, which can be divided into three types: analytical, simulation based, and hybrid models.

Analytical models rely on equations that represent the dynamics of the worms; these models have the benefit of computational efficiency, meaning that they can be easily scaled to predict the behavior of a network of millions of hosts and therefore provide an efficient way to study a problem. The RCS (random constant spread) model [16] is an example of an analytical model. This model is also called the classic

*This work is supported in part by NSF under contract numbers ANI-9977544, ANI-0136969, ANI-0240477, ECS-0225417, and DARPA under contract number N66002-00-1-8934.

SI (susceptible-infected) model in [8] and used to describe the worm spread through homogeneous random contacts between susceptible and infected hosts. The SIR (susceptible-infected-removed) and SIS (susceptible-infected-susceptible) models [4, 3] add treatment and removal of infected hosts into the SI model. Zou et al. introduced an analytical model called the Two Factor worm model that tries to include the effect of human counter-measures [19]. The main drawback of analytical models is that they often do not take into account many important worm characteristics, such as payload size, transport protocol used, and the different probabilities of infection resulting from different topologies and network conditions.

Packet-level network simulations on the other hand, provide a more realistic and detailed model to represent worm spread. Sharif [11] built worm models into GTNetS [10] and was able to simulate networks having hundreds of thousands of hosts and measure the effect of different network parameters on the worm spread rate. Wagner [18] discusses an efficient simulator for worm propagation implemented in the Perl scripting language where he used models for large groups of nodes for the Internet rather than single hosts to reduce complexity as well as simplified models of UDP and TCP behavior. This enabled large simulations but without packet-level detail.

Hybrid models incorporate analytical models and packet-level simulations as well. Liljenstam et al. [5] used the SSFNet [14] simulator with packet-level details for a small section of the network and represented the rest of the Internet with an analytic model.

The study of worst case scenarios has received high interest from researchers; in [16] the authors introduced some improvements to random scanning worms such as Permutation scanning, hit-list scanning, and Flash worm.

The Permutation scanning approach addresses the limitation that in a random scan many addresses are probed multiple times, leading to unnecessary re-infection attempts. In a permutation scan all worms share a common pseudo random permutation of the IP address space. Any infected machine starts scanning just after its point in the permutation, working its way through the permutation, looking for vulnerable machines. Whenever the worm sees an already infected machine (infected machines responds differently to a scan), it chooses a new, random start point and proceeds from there.

In the hit-list scanning approach the worm author collects a list of potentially vulnerable machines before the worm is released. The worm, when released onto an initial machine on this hit-list, begins scanning down the list. When it infects a machine, it divides the hit-list in half, communicating half to the recipient worm, keeping the other half.

A variation on the hit-list scanning approach is the Flash Worm, which employs hit-list scanning with a large scale list containing all the vulnerable hosts on the Internet. In [15] they show that such a worm can infect 95% of a population of one million hosts in 510 milliseconds.

This paper introduces the CFlash (Compact Flash) worm that is a variant of the Flash worm. In this approach the worm does not send the actual list of addresses to its children; rather it sends a smaller version of the address list represented by relative offsets of the addresses to each other.

The remainder of the paper is organized as follows. Section 2 provides some related work and a description of the Flash Worm. Section 3 describes the CFlash worm and its implementation in GTNetS. Section 4 gives a basis for the evaluation of the Flash and CFlash worms as implemented in GTNetS. In Section 5 the results are provided, Section 6 gives the conclusion and future work.

2 Related Work

The Flash Worm was first introduced in [16]. In that work the authors described *hit-list* scanning in the following manner: “Before the worm is released, the worm author collects a list of say 10,000 to 50,000 potentially vulnerable machines, ideally ones with good network connections. The worm, when released onto an initial machine on this hit-list, begins scanning down the list. When it infects a machine, it divides the hit-list in half, communicating half to the recipient worm, keeping the other half.”

The hit-list can be generated using one or several of the following techniques:

- *Stealthy scans.* A fast scan of the whole Internet would be unlikely to attract attention. However, for attackers wishing to be very careful, a randomized stealthy scan taking several months would not be detected.
- *Distributed scans.* An attacker can scan the Internet using a few already-compromised "zombies."
- *DNS searches.* A list of domains can be assembled and the DNS can then be searched for the IP addresses of mail servers or web servers.
- *Spiders.* Web-crawling techniques are similar to search engines and they are used in order to produce a list of most Internet-connected web sites.
- *Public surveys.* For many potential targets, there might be surveys available listing them, such as the Netcraft survey [9].
- *Just listen.* Some applications, such as peer-to-peer networks, wind up advertising many of their servers. Similarly, many previous worms effectively broadcast that the infected machine is vulnerable to further attacks.

The Flash worm works by forming a hit-list that contains all the vulnerable hosts in the Internet. The mechanics of splitting the list with every child can be represented by a logical tree (Figure 1), where a node at each level infects its descendants and divides the list of vulnerable IP addresses among them. The logical tree can be k-way (meaning each instance infects k other hosts). The number of generations (layers in the tree) to infect N vulnerable hosts is $O(\log_k N)$. The total infection time is bounded by this number multiplied by the time required to infect a generation. The authors observe that a Flash worm that has a hit-list of most servers with the relevant service open to the Internet in advance of the release of the worm, appears able to infect almost all vulnerable servers on the Internet in less than 30 seconds.

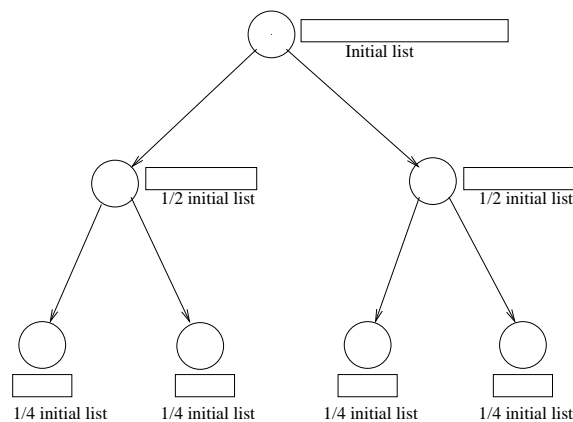


Figure 1: The logical tree for the Flash worm

In a later work [15] the authors revisited their calculations and concluded that to infect 95% of one million host topology it only takes 510 milliseconds. The parameters used for that speed were:

- The logical tree is 3 layers with 9260 secondary nodes infecting 107 addresses each for a total of 1000080.
- The root node is chosen to have high bandwidth (750 Mbps.)

- The distribution of link delays is calculated from the round-trip (RTT) measurements in CAIDA's skitter datasets [2]

The authors also note that one of the main drawbacks of the Flash worm is the lack of robustness, if the list of vulnerable addresses is imperfect. Since it is assembled in advance, and networks constantly change, the list is likely to be out of date by the time the worm is released. This has two effects; some vulnerable hosts may not be on the list which means that the worm will not reach full saturation. More seriously, some addresses may turn to be invulnerable to the worm and if such hosts are on the base of the tree they would prevent all the addresses under them from getting infected and this would be more serious in deep trees. The authors in [15] studied this problem and suggested solutions to deal with it.

3 Compact Flash

The list of vulnerable IP addresses across the Internet can be huge. For instance, according to the netcraft survey for May 2005 there are about 43 Million Apache web servers and 12 Million Microsoft web servers [9]. This means that in the case of a 0-day vulnerability assuming that over 80% will be vulnerable, there will be about 35 Million vulnerable hosts for the Apache servers. The size of the list will therefore be 140 Mbytes (4 bytes per address). Even if the root node had the assumed high bandwidth of 750 Mbps, it would require at least 1.5 seconds just to transfer that list to the first layer children. With that large amount of traffic the probability of being noticed is increased, therefore for an attacker who wishes to achieve optimum speed it is very important to reduce the size of that list as much as possible. By using the CFlash (Compact Flash) approach it is possible to reduce that size by a factor of 1/4 to 1/2, meaning that the size would then be 35 to 70 Mbytes.

The CFlash worm has the same structure as the Flash worm except that the list of IP addresses is represented by the relative offsets rather than the actual IP addresses. It can be argued that if the offsets are small enough (less than 256), then the list can be reduced to a quarter of its original size, as each IP address takes four bytes of storage, while the small offset can take one byte. However, this is not a very practical assumption as there are certain to be two consecutive IP addresses that are separated by more than one byte offset. To deal with this problem two special characters are defined: "two-byte-offset" and "four-byte-offset" and assigned the values 0xff and 0xfe to them.

When the CFlash worm receives the list representing the vulnerable hosts, it calculates the next victim by adding the first offset to its own IP address and takes a chunk of the offset list and sends it to that new victim. In case it encounters 0xff or 0xfe it skips that character and reads the next two or four bytes to determine the correct offset. The size of the chunk to send to the new victim depends on the number of children for that worm (the fan out for the logical tree).

The size of the vulnerable hosts list is now reduced by a factor of 1/2 to 1/4; however, this does not necessarily increase the worm speed by an order of two or four. The reason being that the worm speed is not only dependent on the packet size but on link delay as well. The worm speed (for one generation) is proportional to the following:

$$Speed \propto \frac{1}{packet\ size / BW + link\ delays}$$

This imposes an upper bound on the increase of the speed by reducing the packet size to be $1/link\ delays$. Moreover, if the link delays are much more than the $packet\ size / BW$, then the $link\ delays$ term becomes the dominant term and reducing the packet size improvement is small as will be shown in section 5

The algorithm for the CFlash processing is as follows:

- Initialize *current_IP* = our own IP
- Initialize *victim_IP* = our own IP
- The worm receives the list of offsets representing the vulnerable IP addresses
 - Read the size of the main list
 - *child_list_size* = main list size / number of children
 - for *i* = 0:number of children
 - * copy the i^{th} *child_list_size* from the main list into the i^{th} child list
 - * *victim_IP* = *current_IP* + i^{th} offset
 - * update the *current_IP* by adding up all the offsets in the child list
 - * Send the child list along with its size to the *victim_IP*

3.1 Resilience to Imperfect Maps

One of the main drawbacks of the Flash worm technique is that the initial list of the vulnerable hosts' IP addresses has a good chance of being inaccurate, because the formation of that list is done before the worm starts spreading and the state of the vulnerable population could have changed by the time the worm starts its spread. This can happen in one of the following ways:

- Removal or Patching of vulnerable hosts from the network.
- False positives due to the use of some worm defenses like TCP-ACK [1].
- Some hosts might have dynamic IP addresses associated with them.

The authors of the Flash worm [15] studied this problem in detail and presented two solutions. The first solution is to have a form of an acknowledgement from the child to its parent by adding the address of the parent to the child list. If that acknowledgement is not received after a certain period of time another child is chosen from the child list and another packet is sent to the new host with the same child list. This clearly would complicate the code for the worm and would require too many timers and data structures to handle high number of children cases and to keep state information. The second approach is to add redundancy in infecting hosts, meaning that the child list is sent to more than one host in the list. Figure 2a shows the case of a binary tree, where at each level, a node infects its own two descendants, and then sends worm copies to the two descendants of its sibling, just in case the sibling turned out to be invulnerable and the sibling does the same in reverse. The effect of this is to make it less likely that a portion of the tree will fail.

This approach has the problem that now the list has to be carried on for two layers rather than just one in the original design, because now each node has to know the addresses of the descendants of its siblings and also has to send the proper address list to those descendants. However the same idea can be applied in a more efficient manner, Figure 2b shows the same case of the binary tree but now, at each level, a node infects its own two descendants as well as the next address in each of its descendant's list. The effect of this would still make it less likely that a portion of the tree will fail, as now both chosen hosts (the original descendant and its first descendant) have to be invulnerable for the nodes under them not to get infected.

If we assume that the vulnerability ratio is p then in our simple 3 layer tree model for a leaf node to be infected, the leaf node and its parent should be vulnerable. The probability for a leaf node to be infected is thus p^2 (p raised to the power of the number of layers - 1). This shows that for a list that is 50% accurate only 25% of it will get infected, meaning that 25% vulnerable hosts will not get infected, which is actually 50% of the vulnerable hosts.

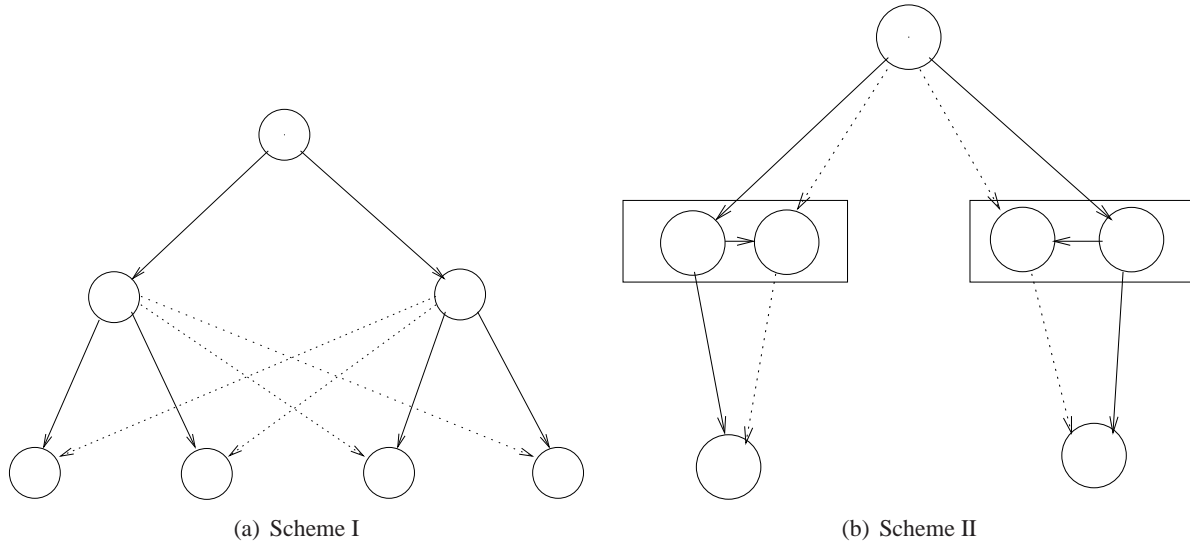


Figure 2: Two schemes for doubling up worm delivery resilience

After using double infection approach (sending the same child’s vulnerable list to two children instead of one). This added redundancy causes the worm to be more resilient to imperfect maps due to the fact that now for a certain node in the list to be not infected the two chosen parents has to be invulnerable, which has less probability of occurring. In this case for a leaf node to get infected either one of its parents could be vulnerable ($2p(1 - p)$) or both of them are vulnerable (p^2) and the leaf node has to be vulnerable as well, thus now the probability of a leaf node to get infected is $2p^2(1 - p) + p^3$ after expansion it would be $2p^2 - p^3$. In that case if the list was 50% accurate then the infection would be 37.5%, meaning that only 12.5% vulnerable hosts from the original list will not get infected (25% of the vulnerable hosts will not get infected.)

3.2 Simulation Model

The Flash and CFlash worms are modeled using GTNetS as application layer objects with the following parameters:

- *Layer4proto*, the layer 4 protocol object to be associated with the worm (either UDP or TCP.)
- *Fanout*, the maximum number of children for each parent node in the logical tree.
- *Vulhosts*, the list of vulnerable IP addresses known to this worm.
- *Infected*, a flag indicating if this node is infected or not.
- *Vulnerable*, a flag indicating if this node is vulnerable to the worm attack or not.
- *Node*, the attached host node.
- *InfectionPort*, the port which has the vulnerable application running on it.
- *TotalInfected*, the number of infected hosts.
- *TotalVulnerable*, the number of vulnerable hosts.
- *PayloadSize*, the size of the infectious payload.

The processing of the flash worm is in the following manner:

- The worm receives the list of vulnerable hosts' IP addresses
 - Read the size of the main list
 - $child_list_size = \text{main list size} / \text{number of children}$
 - for $i = 0:\text{number of children}$
 - * copy the i^{th} $child_list_size$ from the main list into the list of the i^{th} child
 - * $victim_IP = \text{first IP in the child list}$
 - * Send the child list along with its size to the $victim_IP$

4 Experimental Setup

The experiments were conducted using the random tree topology [11]. The random tree is a tree topology with the addition of a random probability factor that determines if a child node should be created or not, which leads to the presence of holes in the IP address space, leading to a more realistic representation of the network.

In this work there is a difference between the logical tree and the physical tree. The logical tree is the tree representing the hierarchy of infection from parent to child nodes regardless of how they are actually connected in the network, while the physical tree is the chosen topology structure to connect the hosts with actual links.

The logical topology for the Flash and CFlash worms is a three-layer tree with a maximum of 1000 children under each parent node. The *layer4proto* is set to UDP, and the *PayloadSize* is set to 400 bytes. The basic setup for all the experiments is that the topology is generated with different parameters setting (bandwidth of links, time delay, and vulnerability) and we start the worm infection and then measure the spread rate.

The physical topology for the experiments consisted of 24 random trees connected together with a ring. Each tree has an address space of 65k and an average of 45k real nodes; the total topology size is thus about 1 million leaf nodes. The size of the list of IP addresses for the Flash worm is 4.119 Mbyte, by using the offsets technique the size becomes 1.03 Mbyte. The bandwidth of the connecting ring links is fixed at 1 Gbps

5 Results

5.1 Effect of Varying Bandwidth

In this section the effect of varying bandwidth of the links in the trees is examined. For the high bandwidth case (Figure 3a) the experiments show that for a one million node topology the Flash worm took 340 milliseconds to infect 95% and 413 milliseconds for full infection, while the CFlash worm took 290 milliseconds to infect 95% and 380 milliseconds for full infection. That is a difference of 30-50 milliseconds is observed due to the reduction of the vulnerable IP list size by using the CFlash worm. In that case small improvement is observed. The high bandwidth caused the $packet_size/BW$ term to be negligible compared to the *linkdelays* term.

The same experiment was carried on low bandwidth links as shown in Figure 3b and it can be observed in that case that the speed difference between the Flash and the CFlash worms is much bigger (about 3 seconds difference for total infection). The CFlash worm is more than 3 times faster than the Flash worm in that case.

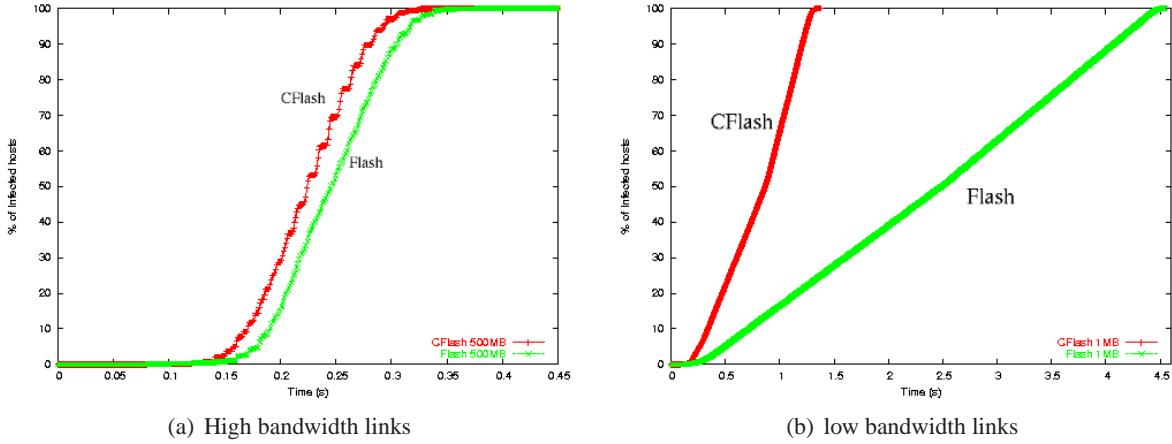


Figure 3: Worm spread for the Flash and CFlash worms

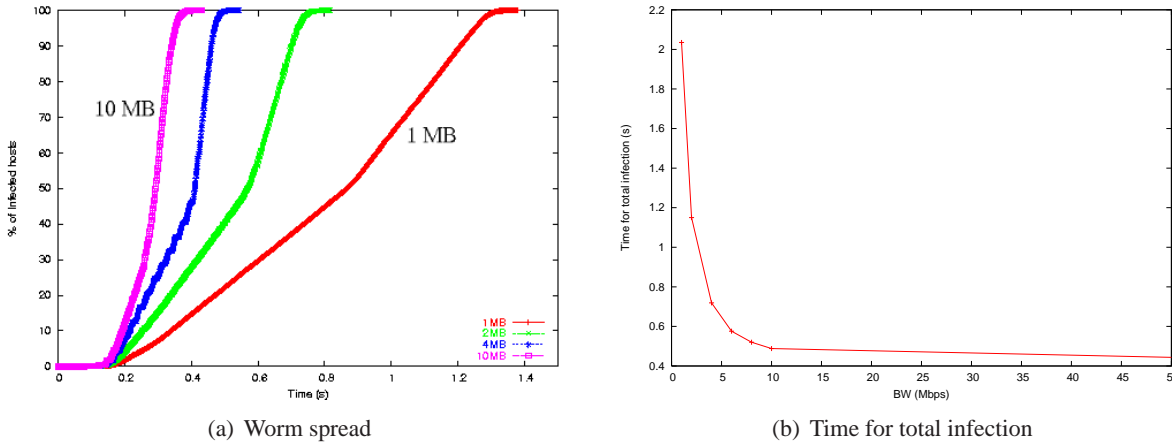


Figure 4: Effect of varying bandwidth of links on the worm spread and infection time

Figure 4 shows the effect of having different bandwidth setting for the connecting links in the trees for the CFlash worm, with fixed time-dealy of 10ms. It is clear that the slope of the infection curve is reduced with reduction of available bandwidth, this is probably due to saturation of the links which is clearer for low bandwidth as the worm spread is not exponential anymore and it tends to be linear.

5.2 Effect of Varying Link-delays

Another set of experiments shows the effect of varying the link delays on the speedup for the CFlash worm relative to the Flash worm, Figure 5 shows the Worm spread for the Flash and Compact Flash worms for different time-delay links with 10MB bandwidth of connecting links. It can be observed that the increase in speed is higher in case of the low time delay of links compared to when the time delay of links is high,

Figure 6 shows the effect of having different time delay setting for the connecting links in the trees on the worm spread and infection time for the CFlash worm with fixed bandwidth of 10MB. It is clear that the slope of the infection curve remains the same but the curve shifts to the right with increase in the time delay of the links.

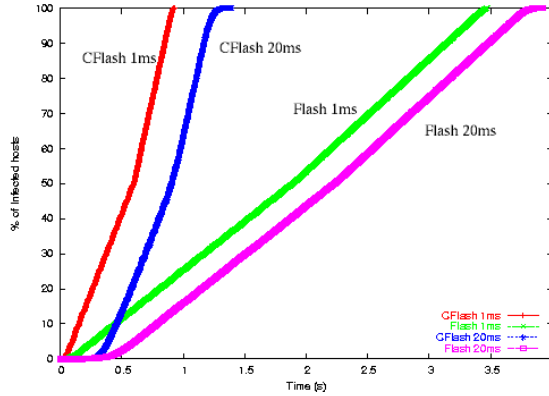


Figure 5: Worm spread for the Flash and Compact Flash worms for different time-delay links

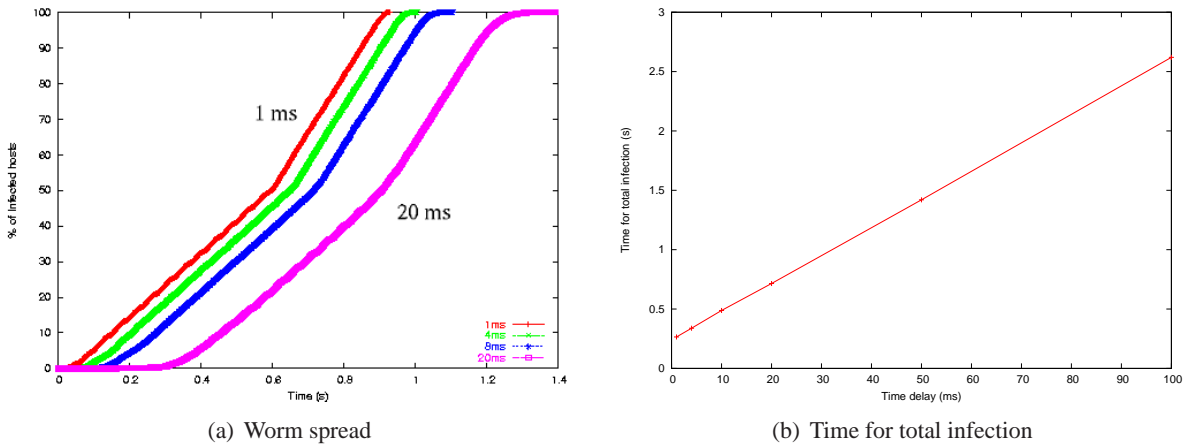


Figure 6: Effect of varying time-delay of links

5.3 Resilience to Imperfect Maps

In this section the effect of having inaccurate map is shown.

Figure 7 shows the simulation results for 20 experiments using different values for the vulnerability ratio and measuring the infection percentage for the original case with single infection. The same figure also shows the effect of the added redundancy by using double infections.

It is clear that when the invulnerability ratio is high the worm spread fails to infect all the vulnerable population even with 50% vulnerability we see that only 30% of the vulnerable hosts (30% of 50% of 1 million = 150k out of 500k vulnerable hosts) are infected. In these tests if the tree was a deep tree (less number of children and more layers), then the effect of the map being inaccurate would be more severe meaning that even with slight inaccuracy the infection would fail to spread to most of the vulnerable hosts.

6 Conclusion and Future Work

It has been shown that by efficiently decreasing the size of the packets sent by the Flash worm, the worm can be sped up further, leading to the possibility of infection of a one million host population in less than 400 milliseconds for high bandwidth links. The improvement is clearer in the low bandwidth case where the CFlash worm becomes 3 times faster than the Flash worm. This main idea can be extended by using compression and decompression of the address list at the infected hosts. A detailed study of the benefit of

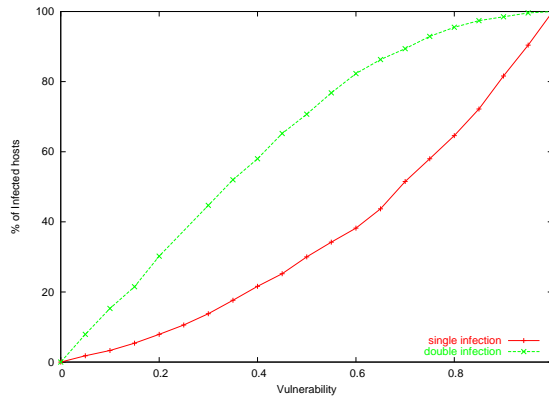


Figure 7: Varying the vulnerability of hosts in the initial list using single and double infections

reduced size against the cost of increased complexity needs to be carried out to determine the optimum size of the infection packets.

This high speed of infection shows that no human counter-measure can stop the worm in time, and even automatic mitigation techniques with time delays over 400 milliseconds are not efficient in stopping that worm once it starts spreading. The only hope of countering such an attack is to stop this kind of worm in the scanning phase before it starts to spread.

Several areas need more careful consideration:

- The effect of having a more realistic topology representing the Internet.
- The effect of containment strategies attempting to delay or stop the worm spread.
- The effect of background traffic.
- Studying real worm outbreak cases and comparing the effect of having a CFlash worm instead in those cases.

References

- [1] M. Abdelhafez and G. Riley. Evaluation of worm containment algorithms and their effect on legitimate traffic. In *IWIA'05: Third IEEE International Workshop on Information Assurance*, pages 33–42, March 2005.
- [2] CAIDA. Skitter datasets. <http://www.caida.org/tools/measurement/skitter>.
- [3] H. W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, October 2000.
- [4] J. O. Kephart and S. R. White. Measuring and modeling computer virus prevalence. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, page 2, Washington, DC, USA, 1993.
- [5] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *WORM'03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 24–33. ACM Press, 2003.
- [6] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Magazine of Security and privacy*, 1(4):33–39, July/August 2003.

- [7] D. Moore, C. Shannon, and J. Brown. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings Internet Measurement Workshop (IMW)*, Marseille, France, November 2002.
- [8] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings IEEE INFOCOM*, San Francisco, CA, USA, March 2003.
- [9] Netcraft. Netcraft web server survey. <http://www.netcraft.com/survey>.
- [10] G. F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12. ACM Press, 2003.
- [11] G. F. Riley, M. I. Sharif, and W. Lee. Simulating internet worms. In *Proceedings of The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pages 268–274, Volendam, The Netherlands, October 2004.
- [12] C. N. Robert Lemos. Msblast epidemic far larger than believed. http://news.com.com/MSBlast+epidemic+far+larger+than+believed/2100-7349_3-5184439.html, April 2004.
- [13] C. Shannon and D. Moore. The spread of the witty worm. Technical report, CAIDA, April 2004.
- [14] SSFNet. <http://www.ssfnet.org>.
- [15] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 33–42. ACM Press, 2004.
- [16] V. P. Stuart Staniford and N. Weaver. How to Own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, USA, August 2002.
- [17] J. Swartz. Cops take a bite, or maybe a nibble, out of cybercrime. *USA TODAY*, September 2003.
- [18] A. Wagner, T. D. bendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *WORM'03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 34–41. ACM Press, 2003.
- [19] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 190–199. ACM Press, 2003.